Take your VNF to the Gym: A Testing Framework for Automated NFV Performance Benchmarking

Raphael Vicente Rosa, Claudio Bertoldo, Christian Esteve Rothenberg

***Abstract*—A Virtualized Network Function (VNF) is a software entity to be run in diverse execution environments with variable configuration options and capabilities (e.g., hardware acceleration) impacting performance. Network Function Virtualization (NFV) resource multiplexed infrastructures can impose hard-to-predict relationships between VNF performance metrics (e.g., latency, frame loss), the underlying allocated resources (e.g. units of vCPU), and the overall system workload. Characterized by many-fold platform configuration and environment variables, the evolving scenario of NFV calls for adequate testing methodologies embracing modern continuous development and integration practices and leveraging open source tools and mindset. To this end, we introduce Gym as our proposed testing framework and methodology for automated NFV performance benchmarking. We present our design principles and the outcomes from a practical validation on a vIMS scenario. A discussion on the lessons learned and the overall NFV performance testing landscape are further contributions of this article.**

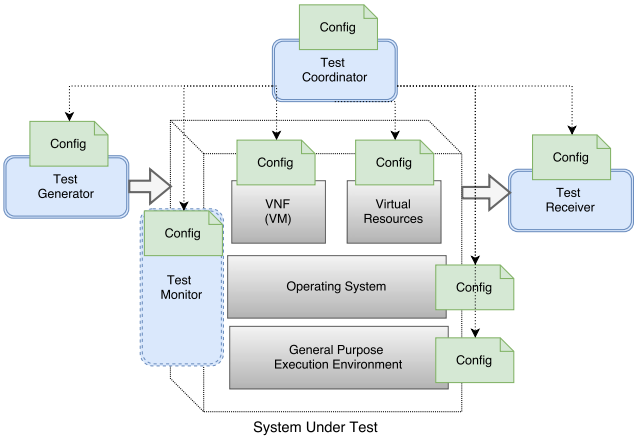


Figure 1. VNF under test scenario illustrating the multiple configuration knobs and the diverse multiple system and platform variables involved.

I. Introduction

As Network Function Virtualization (NFV) matures through the realization of proof of concept implementations, identified challenges towards wider roll-outs include the need of carrier-grade testing and operational standards to match service-continuity and performance predictability levels of current physical infrastructures [1], [2]. Being pure software entities, VNFs lend themselves for continuous deployment and integration following agile DevOps methodologies. As illustrated in Fig. 2, software-oriented processes applied to NFV call for automated testing practices spanning platform portability, functional correctness, and performance benchmarking for each candidate VNF version before turning it available for deployment. A single line of code change passing all functional tests could also undermine the VNF performance for specific workloads and platforms —a risk that calls for standardized testing methods [3], [4], [5], [6] towards adequate VNF benchmarks (e.g., [7], [8], [9]).

The heterogeneity of NFV Infrastructure (NFVI) environments include diverse virtualization options and system capabilities (e.g., HW offloading, kernel bypassing) for varying workloads and diverse resource sharing conditions (e.g., co-located VNFs trashing shared CPU memory caches) [1]. Such a multi-dimensional testing landscape with multiple configuration knobs (see Fig. 1) introduces unprecedented challenges towards useful performance profiles delivering valuable assessments for different stakeholders at different stages, e.g., during VNF development, for pre-deployment NFVI validation, or even for Service Level Agreement (SLA) compliance at runtime.

In our initial work on VNF Benchmarking as a Service (VBaaS) [3], we introduced the problem statement of VNF benchmarking based on "trust, but verify" principles in seek of standardized performance testing allowing proper evaluation of candidate platforms and locations to host (chains of) VNFs with respect to target Key Performance Indicator (KPI)s. In this article, we revisit our functional and architectural vision of VBaaS based on the prototype development and practical evaluation of Gym, the proposed testing framework that allows automated performance benchmarks of NFV embodiments. We advocate for a framework that defines a minimum set of standardized interfaces while allowing user-defined tests along a catalogue of reusable VNF testing procedures and reports with wide- and well-defined system configuration descriptors, workload parametrization (linking to specific traffic generation tools and their parameters), KPI computation, along all supporting code and data expected from a standardized and reproducible benchmarking methodology.

Outcomes of automated performance tests can be used as inputs of Network Function Virtualization Orchestrator (NFVO) embedding algorithms (cf. [4]) and/or parameters to support business decisions such as pricing and allocation of resources to fulfill SLAs. As noted by the vision behind NFV-VITAL [9], standardized characterization of VNF performance enables analyzing optimal sizing and configuration of VNFs in order to automatically:

1) For a given resource configuration, estimate the VNF

The authors are with School of Electrical and Computer Engineering (FEEC), University of Campinas (UNICAMP), Brazil

http://nfvwiki.etsi.org/index.php?title=PoCs_Overview - Accessed on 2017-06-01

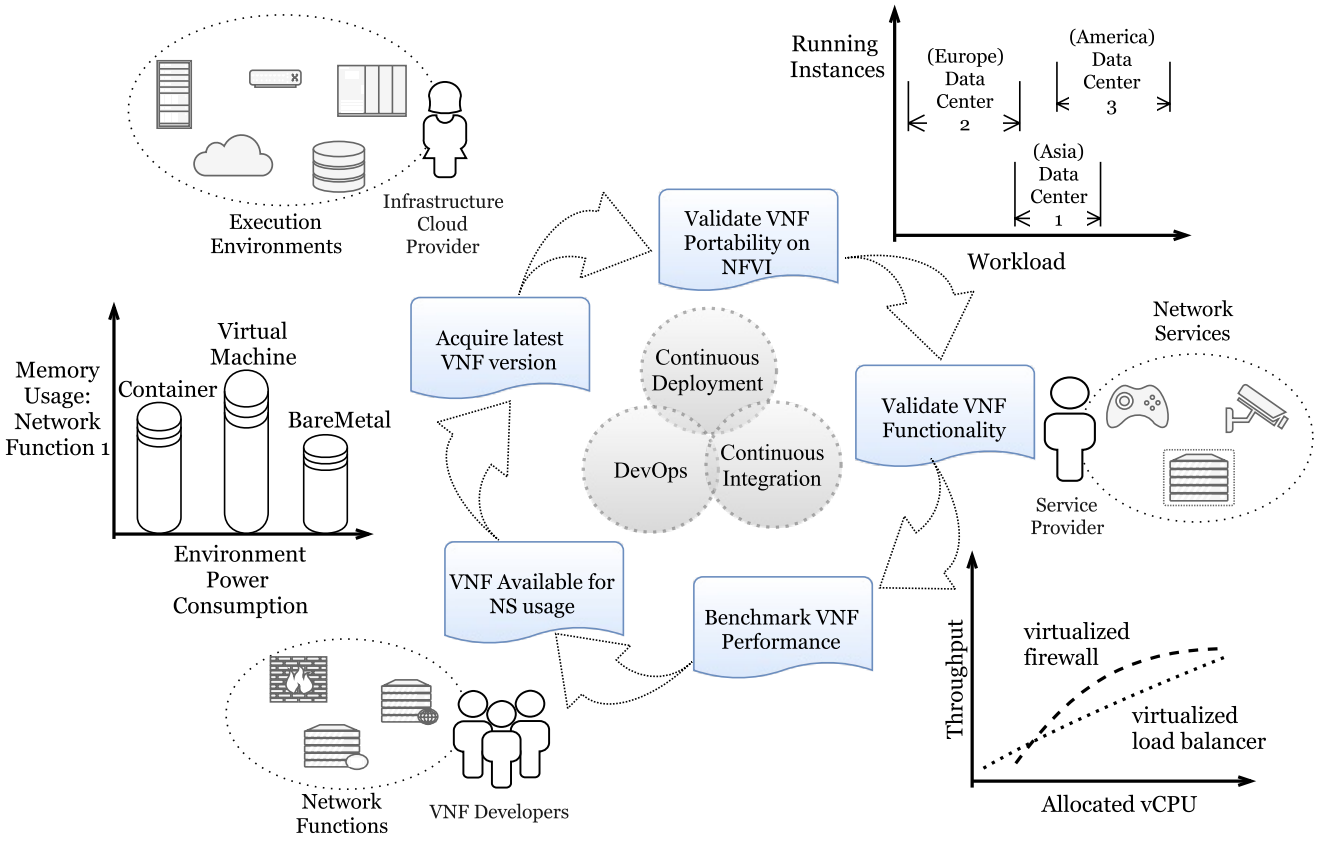


Figure 2. Gym Motivation: Big picture of VNF benchmarks as part of rapid service processes though automation, regression and performance testing.

capacity.

2) For a given workload, determine optimal resource configuration.
3) Evaluate different OS virtualization / HW alternatives and compute system overhead associated to dynamic scaling (up/out/down/in).
4) Fine-tune VNF implementation and performance debugging (i.e. "if you cannot measure it, you cannot improve it" – William Thomson, known as Lord Kelvin).

The article is organized as follows. Section 2 presents the approach and design of the Gym framework and presents a generic workflow to illustrate the main functionalities. Section 3 puts Gym into practice by performing benchmarking tests on an open source IP Multimedia Subsystem (IMS) implementation. Section 4 is devoted to an open discussion on the achievements and limitations, considering aspects from the vIMS experiments as well as more general aspects and challenges on VNF testing framework design and implementation. Section 5 discusses the vibrant related work before the final remarks and conclusions of Section 6.

II. Gym: from Design to Implementation

Taking roots in the former design efforts of VBaaS [3], our early envisioned abstractions evolved into the framework implementation, baptized as **Gym**. Our approach is based on the development of a skeleton of software components delivering the abstractions and tool set in support of practical methodologies to validate, benchmark, and dimension VNFs [6]. Gym is mainly characterized by:

- Modular architecture with stand-alone programmable components
- Simple messaging system following generic Remote Procedure Call (RPC) guidelines
- Extensible set of testing tools and target metrics
- Rich test definition through dynamic compositions of modules
- And flexible methods for output processing and results visualization.

As shown in Fig. 2, Gym aims at introducing new opportunities to different NFV actors. VNF developers can rely on the framework to add automated, repeatable VNFs performance profiling to their agile Continuous Integration and DevOps practices. Service Providers might enhance offered Quality of Service (QoS) with tested-deployed scenarios (e.g., varying workloads in multiple sites), containing transparent sets of operational VNF metrics, targeting Continuous Deployment. Cloud/Infrastructure Providers, when extensively testing VNFs in their execution environments, can use Gym to implement SLA compliance methods to increase the infrastructure reliability and operational efficiency (e.g. energy consumption).

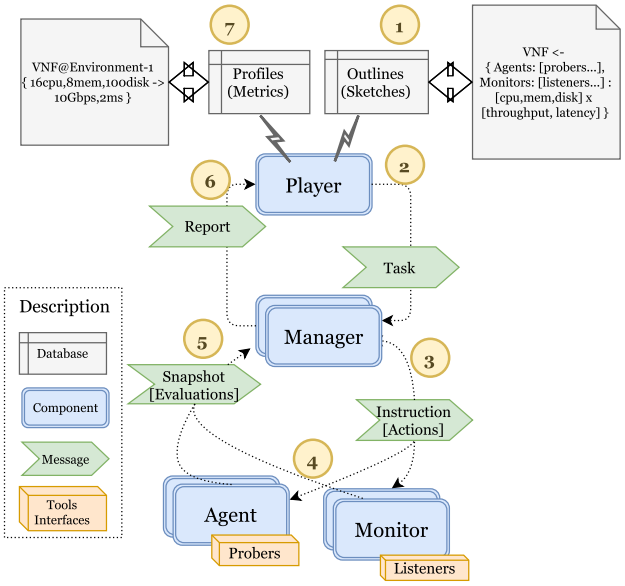


Figure 3. The Gym architecture is based on four main components (Agent, Monitor, Manager, Player), allowing flexible workflows and embodiments as illustrated by the various message exchanges, interfaces, databases, and tools.

A. Conceptual Ideas and Guiding Principles

Design for modularity is one of the main guiding principles of Gym to allow independent software components to be orchestrated on-demand based on well-defined testing objectives without compromising customization and overall extensibility. To address the heterogeneous and complex set of requirements and capabilities of NFV instantiations, the framework offers a high degree of freedom through user-defined composition of sets of tools and evaluation models using simple description formats. Gym overall principles, enunciated below, will come later into further discussion when evaluating a VNF benchmarking use case. The proposed guiding principles to design and build a performance testing framework can be compound in multiple practical ways for multiple VNF testing purposes.

- Comparability: Output of tests shall be simple to understand and process, in a human-readable format, coherent and easily reusable (e.g., inputs for analytic applications).
- Repeatability: Test setup shall be comprehensively defined through a flexible design model that can be interpreted and executed by the testing platform repeatedly but supporting customization.
- Configurability: Open interfaces and extensible messaging models between components for flexible composition of tests descriptors and platform configurations.
- Interoperability: Tests shall be ported to different environments using lightweight components.

B. Architecture

The system architecture of Gym is illustrated in Fig. 3 and comprises the following four main modules:

Agent. Provides extensible interfaces for testing tools (e.g., iperf, ping), named *probers*, to create stimulus in order to collect network and host performance metrics. Agents enable both local (e.g., CPU and disk I/O benchmarks) and distributed (e.g., end-to-end latency/throughput between Agents) measurements, and expose modular APIs for flexible extensibility (e.g., new probers). Agents receive *instructions* from a Manager defining sets of *actions* to consistently configure and run prober instances, parse the results, and send back *snapshots* containing output *evaluations* of the probers' actions.

Monitor. Performs internal and external instrumentation of VNFs and their execution environments in order to extract passive metrics using monitoring tools (e.g., top, tcpdump) interfaces, named *listeners*. Monitors can work jointly with Agents workloads, for instance, when the VNF throughput shall be correlated with the vCPU utilization. Similarly to the Agent, Monitors interact with the Manager by receiving *instructions* and replying with *snapshots*. Different from the generic VNF prober approach of the Agent, Monitors may listen to particular metrics according to capabilities offered by VNFs and their respective execution environment (e.g. CPU cycles of DPDK-enabled processors).

Manager. Responsible for *(i)* keeping a coherent state and consistent coordination of the managed components (Agents and Monitors), their features and activities; *(ii)* interacting with the Player to receive *tasks* and decompose them into a concrete set of *instructions*; and *(iii)* processing *snapshots* along proper aggregation tasks into *reports* back to the Player.

Player. Defines a set of user-oriented, north-bound interfaces abstracting: *(i)* metric extraction descriptors, named *Sketches*, according to the requirements and settings of probers/listeners; and *(ii)* VNF testing *Outlines* containing one or more *sketches* with their configurable parameters. Player might store different *outlines*, and trigger their execution when receiving a testing *Layout* request, that might reference one or more parametrized *outlines*, which are decomposed into a set of *tasks* orchestrated by Managers to obtain the *reports*. Interfaces are provided for storage options (e.g., database, spreadsheets) and visualization of the extracted *reports* into *profiles*.

Two relevant terms deserve further explanation:

Outline: Used as input by the Player module, it defines how to test one or more VNF types following a particular syntax in YAML to express structural settings (e.g., Agents/Monitors topology) and functional properties (e.g., probers/listeners parameters), named *sketches*. *Profile:* Refers to the formatted outcome composed by the outputs of an *outline* execution and the requested Layout scenario. A *Profile* represents a mapping between virtualized resources (e.g., vCPU, memory) in a given environment and VNF performance/benchmarking metrics (e.g., throughput, latency between in/out or ports), abstracting VNF allocation with certain resources to delivering a unifying metric for a given (predictable/measured) performance quality.

C. Messaging System and Workflow

Gym core components communicate through REpresentational State Transfer (REST) Application Programming Interface (API) using generic RPC calls with custom JSON message formats. In the following, we describe a generic workflow based on request-reply message exchanges and pairwise component interactions represented as numbered (1 to 7) circles in Fig. 3.

1) The first step consists of a user defining the composition of the VNF testing *Outline* through *Sketches* containing the structural and functional requirements to express target performance metrics to generate a VNF *Profile*.
2) The Player processes the parametrized *Outline* considering the features offered by the associated Manager(s). The output is a workflow of *tasks*, in sequence or parallel, submitted to a selected Manager that satisfies (i.e. controls a matching set of Agents/Monitors) the *Outline* requirements. Based on input variables, an *Outline* can be decomposed into different sets of *tasks* with the corresponding high-level *probers/listeners* parameters.
3) The Manager decomposes *tasks* into a coherent sequence of *instructions* to be sent to Agents and/or Monitors. Inside each *instruction*, sets of *actions* define parametrized execution procedures of *probers/listeners*. Sequential or parallel *tasks* may include properties to be decomposed into different sets of *instructions*, for instance, when sampling cycles might define their repeated execution.
4) By interpreting *action* into a *prober/listener* execution, an Agent or Monitor performs an active or passive measurement to output metrics via a pluggable tool. A VNF developer can freely create a customized prober or listener to interface her tests and extract particular metrics. An interface of such a tool is automatically discovered by an Agent/Monitor and exposed as "available" to Managers and Players along the corresponding execution parameters and output properties.
5) After computing the required metrics, a set of *evaluations* (i.e. parsed *action* outputs) integrate a so-called *snapshot* sent from an Agent/Monitor to the Manager. A *snapshot* associated to a specific *task* is received from the Agent/Monitor that received the corresponding *instruction*. An *evaluation* contains timestamps and identifiers of the originating prober/listener, whereas a *snapshot* receives an Agent/Monitor unique identifier along the host name information.
6) After processing all the *instructions'* related tree of *snapshots*, the Manager composes a *report*, as a reply to each *task* requested by the Player. The Manager can sample *snapshots* in a diverse set of programmable methods. For instance, a *task* may require cycles of repetition, so the correspondent *snapshots* can be parsed and aggregated in a report through statistical operations (e.g., mean, deviation, confidence intervals).
7) Finally, the Player processes the *report* following the *profile* metrics definition, as established initially during the *outline* decomposition. While the *profile* contains filtered evaluation metrics and parameters, *snapshots* can be aggregated/sampled into a *report*. Results can be exported in different file formats (e.g., csv, json, yaml) or saved into a database for further analysis and visualization. For instance, in our current Gym prototype we integrate two popular open source components, the Elasticsearch database and the Kibana visualization platform —tools providing high flexibility in querying, filtering and creation of different visual representations of the extracted *profiles*.

III. A Day in the Gym: vIMS Performance Testing

We now exercise the Gym framework in the case study of IMS telecom network functions [10], in scope of ETSI NFV ISG proof-of-concepts and also target of related work on VNF benchmarking [9]. Understanding vIMS performance for a given resource allocation can be relevant for NFVI configuration and setting adequate system parameters in light of potential price and SLA offerings in multi-vendor cloud infrastructures. Throughout the practical evaluation of the chosen use case, we assess Gym functionalities and further discuss the development of testing methodologies following the architectural design aspects introduced by Gym.

A. Prototype Implementation

Gym provides a framework for VNF testing and features off-the-shelf Linux tools to generate generic metrics. More specifically, the implemented Monitor component includes:
Host listener: A listener process to record resource metrics such as the utilization of CPU, memory, disk and network. Based on *psutil* cross-platform library, and parametrized by interval (sampling rate) and duration, the host listener is able to extract multiple host runtime metrics (approx. 80+).

User-defined, VNF-specific metrics are supported through component extensions. For our vIMS benchmarking purposes, the following prober was added to the deployed Agent.
SIPp prober: A prober integrated to interface the SIPp open source SIP traffic generator used to stress the vIMS deployment based on the four-step SIP registration procedure [11], herein called *transaction*. The Gym *Outline* and *Profile* describing the SiPp prober input parameters and output metrics are shown in Fig. 4 and publicly available.

B. Scenario and Testbed

The vIMS under test is the Clearwater open source project, which provides VM form factors for the different IMS network functions, namely Edge Proxy / P-CSCF ("Bono"), SIP Router I/S-CSCF ("Sprout"), and the HSS Cache ("Homestead"). We opt for OpenStack as the Virtualized Infrastructure Manager (VIM) to deploy and host the vIMS subject of the Gym benchmarking experiments.

Figure 4 shows the experimental scenario featuring a total of five Virtual Machine (VM)s interconnected through a layer 2 network. Gym and vIMS main components constitute a simple topology: a *Monitor* instance runs (as an independent daemon) inside each of the three vIMS main network function VMs; one *Agent* interfaces the *SIPp prober* in another VM; and the *Manager* and *Player* running in the fifth VM. The three Clearwater vIMS network functions (Bono, Sprout and Homestead) run on OpenStack compute nodes based on Linux Ubuntu

'Multi-vendor on-boarding of vIMS on a cloud management framework" http://www.etsi.org/technologies-clusters/technologies/nfv/nfv-poc - Accessed on 2017-06-01

https://pypi.python.org/pypi/psutil - Accessed on 2017-06-01

http://sipp.sourceforge.net - Accessed on 2017-06-01

https://github.com/intrig-unicamp/gym - Accessed on 2017-06-01

http://www.projectclearwater.org/ - Accessed on 2017-06-01

Server 14.04.3 with different default flavors: $m1.small$ (1-vCPU/2GB-RAM), $m1.medium$ (2-vCPUs/4GB-RAM) and $m1.large$ (4-vCPUs/8GB-RAM).

C. Experimental Evaluation

We follow the experimental workflow shown in the methodology description of Fig. 4. For each VM flavor, ten 20 second long benchmarking runs were executed with the Monitor host listener capturing metrics every second. To analyze SIPp prober versus host metrics and derive a benchmarking behavior according to an IMS stress ladder workload [10], the following SIPp *Outline* parameters were defined:

- Transaction rate increase step: 100 transactions/s.
- Interval of transaction rate increase step: two seconds.
- Maximum transaction rate: 1000 transactions/s.
- Maximum simultaneous transactions: 1000.
- Transport protocol: UDP.

Table I summarizes the observed results, with the first grouped columns presenting the overall amount of transactions *Sent*, *Failed* (vIMS could not answer/complete), and *Ack* (completed) by the SIPp prober. Note the existence of *Delayed/Queued* transactions, neither *Ack* nor *Failed*, corresponding to pending events vIMS could not provide an answer during the experiments runtime. In terms of efficiency, Table I presents the amount of transactions *Sent* by the SIPp Prober divided by the transactions that could be completed in theory, as well as vIMS efficiency, representing the amount of completed transactions (*Ack*) over the amount of *Sent*. The observed results point to performance issues when using a $m1.small$ VM configuration.

The explanation behind the systems limits faced by $m1.small$ is presented by the Monitor be observed by the resource metrics collected by the Monitor. Figure 5 presents the overall system CPU percentage usage, mean, and 95% confidence intervals, of each monitored vIMS component. The solid line represents the SIPp prober output in terms of average transaction rate of SIP registration attempts following the "Stressful Ladder" as per the Outline parameters. Note that the SIPp tool follows a congestion-avoidance like behaviour by automatically falling back according to the fail rate.

We can observe that in all three VM flavors, the I/S-CSCF VNF (Sprout) accounts for the largest CPU consumption, as expected from the central signaling function of the IMS core network. In the case of the $m1.small$ VM (Fig. 5(a)) configuration, Sprout suffers from CPU over-consumption and saturates at around 500 transactions/s, whereas $m1.medium$ and $m1.large$ reach round 800 to 900 transactions/s.

We may conclude that, in general, scaling up virtual resources leads to higher vIMS efficiency. However, despite running experiments in an arguably well-controlled environment with low background interference, the metrics present high variability. Curiously, $m1.large$ does not consistently surpass $m1.medium$, as one may expect from a resource-richer configuration. This observation only confirms the practical challenges behind VNF benchmarking.

Table I
EFFICIENCY OF VIMS AND SIPP PROBER PER OPENSTACK FLAVOR

VM Flavor	Transactions			Efficiency	
	Sent	Failed	Ack	SIPp Prober	vIMS
m1.small	5433	206	4227	48%	77%
m1.medium	10924	49	10187	99%	92%
m1.large	10337	232	9170	93%	88%

IV. DISCUSSION

While the presented experimental work features a limited combination of Gym, OpenStack, and vIMS using a simple VM testbed, there are sufficient partial results for a rich discussion and a critical analysis on different aspects.

VNF Testing Challenges. As exemplified by our vIMS benchmarking efforts, designing and implementing a generic VNF testing framework is subject to multiple challenges requiring further investigation:

- Consistency: Naturally, our first insight goes to question if a VNF, when deployed in a certain execution environment, delivers a given performance described in its extracted *Profile*; and, especially, when tested and put in production using multiple virtualization technologies and concurrent system workloads;
- Stability: VNF performance measurements need to present consistent results over different scenarios. Consequently, we would like to answer if test descriptors transparently handle service/resource definitions and metrics of VNFs placed in heterogeneous environments;
- Goodness: A VNF might be tested with different allocated resources and stimuli, unlike the possibilities of production environments. Crucially, we would like to comprehend how well testing results, and associated stimuli, correspond to VNF measured performance when running in execution environments under real workloads.

Resource Optimization and SLAs. Greatly facilitated by an automated approach, our experimental evaluation unveils some patterns that could be useful for optimized configurations by sizing the VMs according to their resource demands (e.g., Sprout $\Rightarrow$ $m1.large$, Bono $\Rightarrow$ $m1.small$). Further optimization and resource allocation strategies (e.g., CPU pinning) should be also investigated to derive more complete scalability recommendations in addition to wider experimentation with realistic workloads, altogether yielding more reliable vIMS performance profiles. We look for a better understanding on how much VNF testing *profiles* could be part of network SLAs. For instance, complementing continuous monitoring, or an optimized process in terms of performance and cost reduction.

Comparability, Repeatability, and Interoperability All results are extracted based on the vIMS *Profile* by the Gym Player. The obtained metrics can be exported in different file formats or committed to a database for comparison purposes. The same *Outline* can be used by Gym to run performance tests in other virtual environments and extract the same type of metrics. This allows Gym users to continuously execute the same pattern of benchmarking in their own deployments with the ability to customize and debug their tests. Gym can be deployed in heterogeneous environments as the main requirements sit on plain Linux and Python support. All

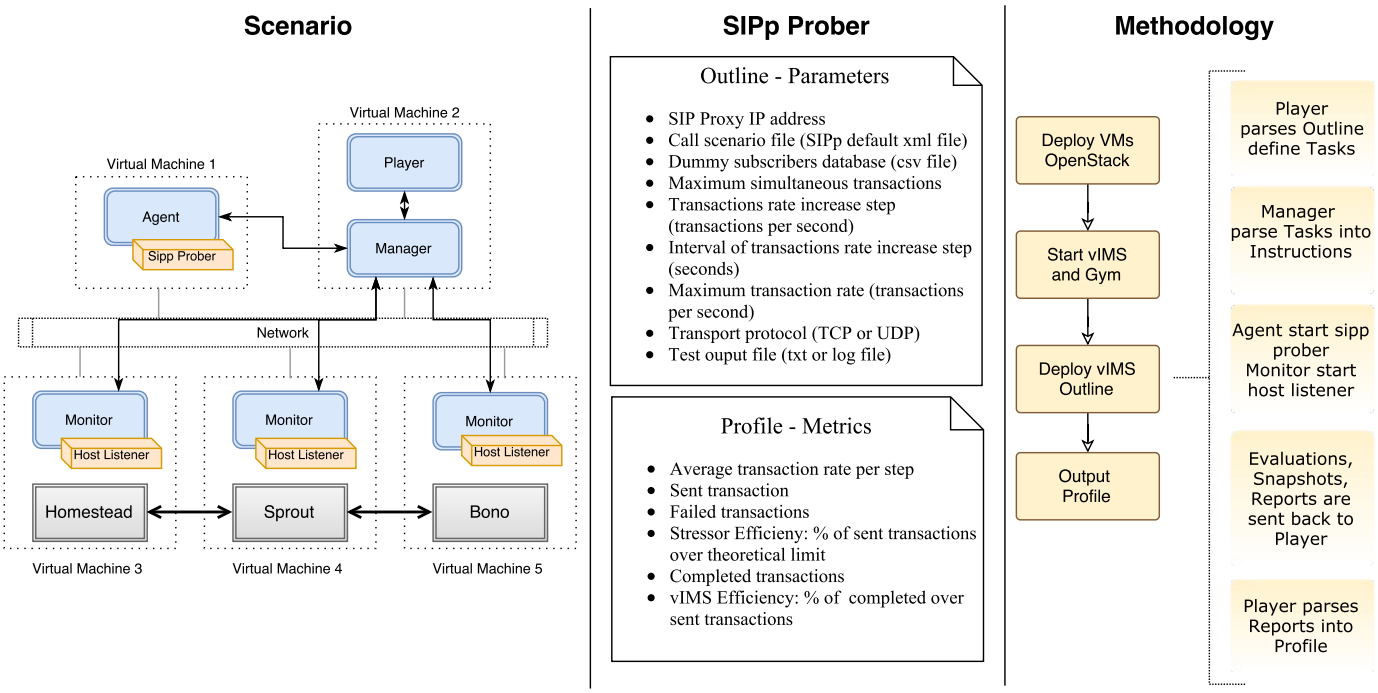


Figure 4. Testbed, SIPp prober outline parameters / profile metrics, and experimental methodology.

components, together with the developed SIPp prober, can be re-used to perform benchmarking tests in case of alternative virtualization (e.g., containers) or bare metal deployments.

Customization and Configurability. Gym provides a skeleton of components well-suited for customized development of arbitrary VNF testing methodologies. Using *Sketches* and *Outline* to benchmark the vIMS offered unfettered choices for customized methodologies based on specific topologies, workloads, metric extractions, and so on. The composition of an *Outline* is a recipe that, when interpreted by the Player component, guides the architectural and functional definitions of VNF tests leveraging Agent/Monitor features. The presented use case exemplifies Gym extensibility by showing a SIPp prober easily integrated to drive the vIMS benchmarks.

NFV Orchestration. In line with our initial VBaaS vision [3], Gym was developed agnostic to any particular NFVO. We envision life cycle management interfaces in Gym to provide workflows for flexible VNF testing. NFVO would consume APIs exposed by Gym to extract desired VNF Profiles and explore them in decision making processes of VNF allocation in terms of target host and resource allocation.

V. Related Work

In line with our initial theoretical vision on VNF benchmarking [3], [7] proposes a structured approach to develop benchmarking methodologies tailored to VNF. The use of performance profiles in support of NFV DevOps workflows has been recently proposed [8] to support management and orchestration decisions leveraging offline profiling of complex service chains.

While developing Gym, we sought alignment [4], [5] with ongoing work at the IETF/IRTF, where under the umbrella of "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure" [12], relevant guidelines are being discussed towards standardized VNF benchmarking. Likewise, Gym was influenced by related efforts at the ETSI ISG NFV Testing Group [6] defining requirements and recommendations for VNFs and NFVI validation. Gym shares similarities to NFV-VITAL [9] with regard to the overall problem statement and framework approach as well as our so-called vIMS efficiency metric, which could be used in auto-scaling strategies after detecting saturation in vIMS transactions per unit of time.

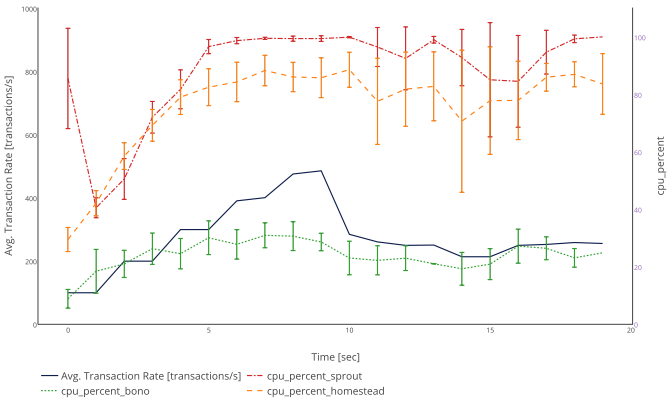
A number of open source projects sprint common abstractions for benchmarking VNFs and the underlying infrastructures. Closest related to Gym, OPNFV incubated projects include *(i)* Yardstick, targeting infrastructure compliance when running VNF applications; *(ii)* QTIP, providing definitions towards platform performance benchmarking; and *(iii)* Bottlenecks, proposing a framework to execute automatic methods of benchmarks to validate VNFs deployment during staging. Compared to Gym, these efforts are very much tied to their choice of technologies, compromising portability and repeatability due to the focus on supporting OPNFV developments without broader aspirations of generic VNF testing tools.

The extensible and modular approach of Gym through Outlines and Profiles allows embracing such standalone projects by integrating them as new probers and listeners along user-defined metrics and testing workflows. One such a candidate open source tool we intend to support in the Gym framework through Agent extensions is NFPA (Network Function Performance Analyzer) [13], which was also born to address the

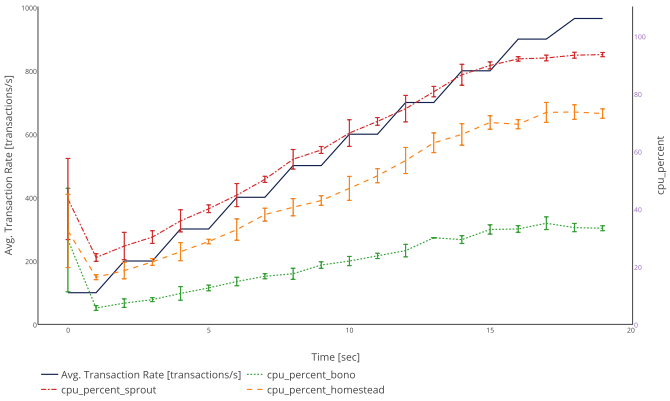
https://wiki.opnfv.org/display/yardstick - Accessed on 2017-06-01

https://wiki.opnfv.org/display/qtip/Platform+Performance+Benchmarking - Accessed on 2017-06-01

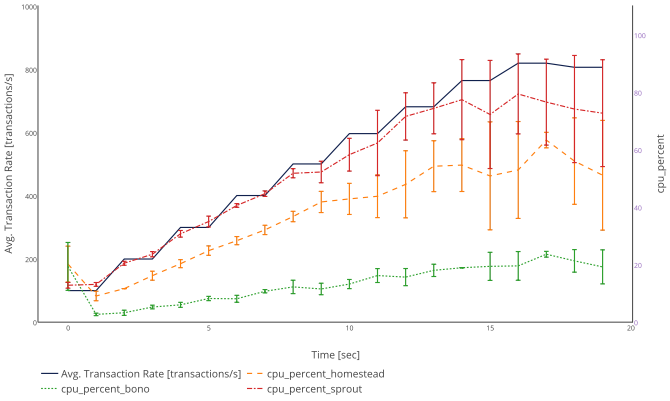
https://wiki.opnfv.org/display/bottlenecks - Accessed on 2017-06-01



(a)



(b)



(c)

Figure 5. Average Transaction Rate (transactions per sec) vs. CPU Usage (%). (a) VNFs over $m1.small$ flavor – Raw data at https://plot.ly/~bertoldo/848. (b) VNFs over $m1.medium$ flavor – Raw data at https://plot.ly/~bertoldo/830. (c) VNFs over $m1.large$ flavor – Raw data at https://plot.ly/~bertoldo/856.

frustrating landscape of benchmarking comparison of network functions over varying SW/HW systems. A last but not least inspiring independent related open source effort is ToDD, which walks in the direction of an on-demand extensible framework for distributed testing of network capacity and connectivity but without focus on NFV or complex workflows.

VI. Conclusions and Future Work

The software nature of VNFs and the multi-dimensional and time-varying aspects of heterogeneous virtualized environments call for adequate methods to assess the infrastructure capabilities with regard to target performance levels. As an evolution of our initial VNF benchmarking vision [3], this article introduced the Gym testing framework along the principles behind our open source implementation.

The vIMS test deployment served as a practical validation of the current Gym prototype, illustrating both its potential and the wider open challenges of automated performance benchmarking in NFV. In spite of the identified limitations, we conclude that Gym offers a meaningful apparatus to express VNF testing abstractions that can be certainly explored in continuous development and integration methodologies. On a recent proof-of-concept evaluation of Gym [14], we used Open vSwitch (OVS) as the VNF under test.

Multiple directions overtake our future work, many of them driven by 5G realization efforts leveraging NFV and Software Defined Networking (SDN) [15]. Examples include benchmarking tests using OpenAirInterface components and NFPA [13] as traffic generator and metric collector. As an architectural framework and an open source project, Gym is still very much in its infancy. We expect Gym to keep evolving, not only in terms of low-level debugging but broadly driven by the community. We envision user-contributed extensions in Gym to support different testing tools, to evaluate newborn VNFs, and to allow users to build and replicate tests by reporting profiles and maintaining common repositories for reproducible research practices involving VNF testing and analytics. Furthermore, we foresee multiple research opportunities when applying a Gym-like approach to NFVs at runtime in support of resource orchestration and business-oriented decisions.

Acknowledgments

This research was partially supported by FAPESP grant #14/18482-4 and by the Innovation Center, Ericsson S.A., Brazil, grant UNI.58.

References

[1] P. Veitch, M. J. McGrath, and V. Bayon, "An Instrumentation and Analytics Framework for Optimal and Robust NFV Deployment," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 126–133, Feb. 2015.

[2] R. Mijumbi *et al.*, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.

[3] R. V. Rosa, C. E. Rothenberg, and R. Szabo, "VBaaS: VNF Benchmark-as-a-Service," in *2015 Fourth European Workshop on Software Defined Networks*, Sept. 2015, pp. 79–84.

https://github.com/toddproject/todd - Accessed on 2017-06-01

http://www.openairinterface.org/ - Accessed on 2017-06-01

[4] ——. (2015) VNF Benchmark-as-a-Service. Internet draft. Accessed on 2017-06-01. [Online]. Available: https://www.ietf.org/archive/id/draft-rorosz-nfvrg-vbaas-00.txt
[5] ——. (2016) VNF Benchmarking Methodology. Internet draft. Accessed on 2017-06-01. [Online]. Available: https://tools.ietf.org/id/draft-rosa-bmwg-vnfbench-00.html
[6] ETSI GS NFV-TST, "ETSI GS NFV-TST 002 V1.1.1 - Report on NFV Interoperability Testing Methodology," Oct. 2016, accessed on 2017-06-01. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-TST/001_099/002/01.01.01_60/gs_NFV-TST002v010101p.pdf
[7] J. Blendin *et al.*, "Towards a Structured Approach to Developing Benchmarks for Virtual Network Functions," in *2016 Fifth European Workshop on Software Defined Networks*, Oct. 2016.
[8] M. Peuster and H. Karl, "Understand your chains: Towards performance profile-based network service management," in *Proceeding of the Fifth European Workshop on Software Defined Networks (EWSDN).*, Oct 2016.
[9] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, "Nfv-vital: A framework for characterizing the performance of virtual network functions," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov 2015, pp. 93–99.
[10] D. Thißen, J. Miguel, and E. Carl, "Evaluating the Performance of an IMS / NGN Deployment," *Proc. 2nd Work. Serv. Platforms, Innov. Res. new Infrastructures Telecommun. SPIRIT*, 2009, accessed on 2017-06-01. [Online]. Available: http://subs.emis.de/LNI/Proceedings/Proceedings154/gi-proc-154-224.pdf
[11] ETSI, "ETSI TS 186 008-2 V2.1.1 - IMS Network Testing," 08 2013, accessed on 2017-06-01. [Online]. Available: http://www.etsi.org/deliver/etsi_ts/186000_186099/18600802/02.01.01_60/ts_18600802v020101p.pdf
[12] A. Morton. (2016) Considerations for benchmarking virtual network functions and their infrastructure. Internet draft. Accessed on 2017-06-01. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-bmwg-virtual-net/
[13] L. Csikor, M. Szalay, B. Sonkoly, and L. Toka, "Nfpa: Network function performance analyzer," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov 2015, pp. 15–17.
[14] R. V. Rosa and C. Rothenberg, "Taking Open vSwitch to the Gym: An Automated Benchmarking Approach," *To appear in IV Workshop pre IETF/IRTF*, Jul. 2017.
[15] D. Kreutz *et al.*, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, p. 63, 2015.

BIOGRAPHIES

Raphael Vicente Rosa (raphaelvrosa@dca.fee.unicamp.br) currently pursues his thesis on multi-domain distributed NFV as a PhD student in University of Campinas, Brazil. During the last two years, he worked as a visiting researcher in Ericsson Research Hungary, where he contributed to EU-FP7 Unify project and developed activities within H2020 5G Exchange project. His main interests sit on state-of-the-art SDN and NFV research topics.

Claudio Bertoldo (bertoldo@dca.fee.unicamp.br) is a fresh post-graduated (M.Sc.) by the University of Campinas, Brazil. He has been involved with next generation fixed and mobile broadband networks since 2007, working at telecommunications companies such as Telefónica and Huawei, and also at several startups. His research interests include Network Functions Virtualization and Next Generation Mobile Networks.

Christian Esteve Rothenberg is an Assistant Professor in the Faculty of Electrical & Computer Engineering (FEEC) at University of Campinas (UNICAMP), Brazil, where he received his Ph.D. and currently leads the Information & Networking Technologies Research & Innovation Group (INTRIG). His research activities span all layers of distributed systems and network architectures and are often carried in collaboration with industry, resulting in multiple open source projects in SDN and NFV among other scientific results.